МИНОБРНАУКИ РОССИИ САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ «ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА) Кафедра МОЭВМ

КУРСОВАЯ РАБОТА

по дисциплине «Алгоритмы и структуры данных»

ТЕМА: АВЛ - ДЕРЕВЬЯ

Студент гр. 8304 Сани З. Б

Преподаватель Фирсов М. А.

Санкт-Петербург

2019

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студент Сани Заяд Бакеко

Группа 8304

Тема работы AVL-деревья

Исходные данные:
Написать программу создание программы для генерации заданий с ответами к ним для проведения текущего контроля среди студентов. Задания и ответы должны выводиться в файл в удобной форме: тексты заданий должны быть готовы для передачи студентам

Содержание пояснительной записки:

- Содержание
- Введение
- AVL-деревья
- Тестирование
- Исходный код
- Использованные источники

Дата выдачи задания: 11.10.2019 Дата
сдачи реферата:

Дата защиты реферата:24.12.2019

Студент Сани З. Б

Преподаватель Фирсов М. А.

АННОТАЦИЯ

В данной работе была создана программа, использующая язык программирования C++, с терминальным интерфейсом, который предоставляет пользователю опции уровня сложности для создания AVLTree, вставки элемента, а также удаления элементов, и он отображает сбалансированное дерево для них в терминале, а также сохраняет его в выходной текстовый файл.

SUMMARY

In this work, a program was created using C++ programming language, with a terminal interface which gives the user options of difficulty level to create a AVLTree ,insert element as well as remove elements and it displays the balanced tree to them in the terminal and as well save it to an output text file.

СОДЕРЖАНИЕ

Введение 5

1. AVL-деревья 6
2. Описание структур данных и функций 8
3. Интерфейс Программы 9
4. Тестирование 11

Заключение 15

Список использованных источников 16

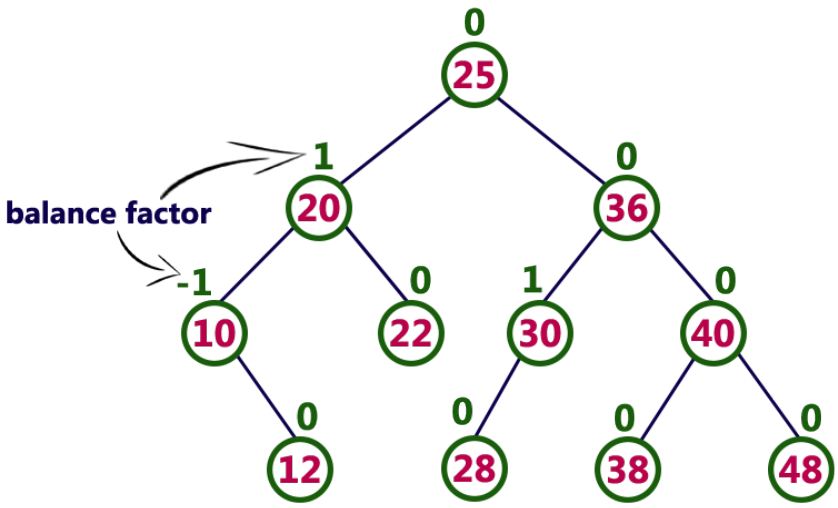
ВВЕДЕНИЕ

Целью данной курсовой работы является реализация программы для проверки построения AVL-деревьев. Дерево AVL-это самобалансирующееся двоичное дерево поиска (BST), в котором разница между высотами левого и правого поддеревьев не может быть больше одного для всех узлов. Дерево AVL гарантирует логарифмический рост высоты дерева позволяющее быстро выполнять основные операции дерева поиска: добавление, удаление и поиск узла.

1. AVL-деревья

Дерево AVL- это самобалансирующееся двоичное дерево поиска (BST), в котором разница между высотами левого и правого поддеревьев не может быть больше одного для всех узлов. Дерево AVL гарантирует логарифмический рост высоты дерева позволяющее быстро выполнять основные операции дерева поиска: добавление, удаление и поиск узла.

Если вставка или удаление элемента приводит к нарушению сбалансированности дерева, то выполняется его балансировка. Коэффициент сбалансированности узла (balance factor) – это разность высот его левого и правого поддеревьев. В АВЛ-дереве коэффициент сбалансированности любого узла принимает значения из множества {-1, 0, 1}.



Пример дерева AVL

После добавления или удаления элемента необходимо обновить коэффициенты баланса родительских узлов. Если какой-либо родительский узел находится вне диапазона {-1,0,1}, поддерево должно быть сбалансировано вращением(Rotation).

Types of Rotation :

1) Правый поворот (RR-rotation, right rotation)
2) Левый поворот (LL-rotation, left rotation)

3) Лево-правый поворот (LR-rotation, left right rotation)

4) Право-левый поворот (RL-rotation, right left rotation)

Когда мы удаляем узел, возникают три возможности.

1) узел, который будет удален, является листом: просто удалите из дерева.

2) узел для удаления имеет только один дочерний элемент: скопируйте дочерний элемент в узел и удалите дочерний элемент

3) узел, подлежащий удалению, имеет два дочерних элемента: найдите inorder преемника (successor) узла. Скопируйте содержимое преемника inorder на узел и удалите преемника inorder. Обратите внимание, что предшественник (predecessor) inorder также может быть использован.

AVL delete Implementation:
Пусть W-узел, подлежащий удалению
1) Выполните стандартное удаление BST для W.

2) начиная с W путешествуйте вверх и найдите первый несбалансированный узел. Пусть Z-первый несбалансированный узел, Y-более высокий дочерний элемент Z, а X - более высокий дочерний элемент

Y. обратите внимание, что определения X и Y отличаются от вставки здесь.

3) повторно сбалансируйте дерево, выполнив соответствующие вращения на поддереве, коренящемся с Z. может быть 4 возможных случая, которые необходимо обработать, поскольку X, Y и Z могут быть организованы в 4 способа. Ниже приведены возможные механизмы 4:

а) Y является левым потомком Z и X является левым потомком y (слева левое дело)(Left Left Case)

б) Y является левым потомком Z и X-это право ребенка y (слева справа случае)(Left Right Case)

c) y-правильный ребенок z, а x-правильный ребенок y (правый правый случай)(Right Right Case)

д) Y-это право ребенка на Z и X является левым потомком г (справа слева случае)(Right Left Case)

После фиксации Z нам, возможно, придется также исправить предков Z.

2 Описание структур данных и функций

Функция main осуществляет всё взаимодействие с пользователем, она приводит к созданию всех остальных используемых структур данных.

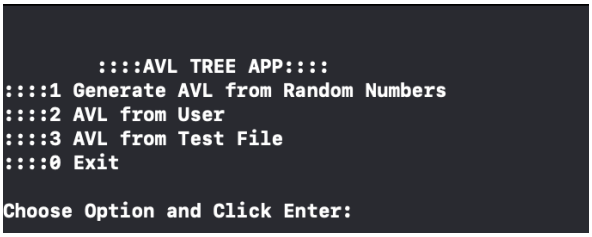
Специально для решения данной задачи был написан класс AVLtree, реализующий AVL дерево с использованием динамической памяти.

Этот класс имеет следующий интерфейс:

- insert - позволяет добавить новое значение в дерево
- remove - позволяет удалить переданное значение, если оно содержится в дереве
- printBracketNotation - возвращает скобочное представление дерева.
- deleteTree - рекурсивно удалит каждый узел в дереве

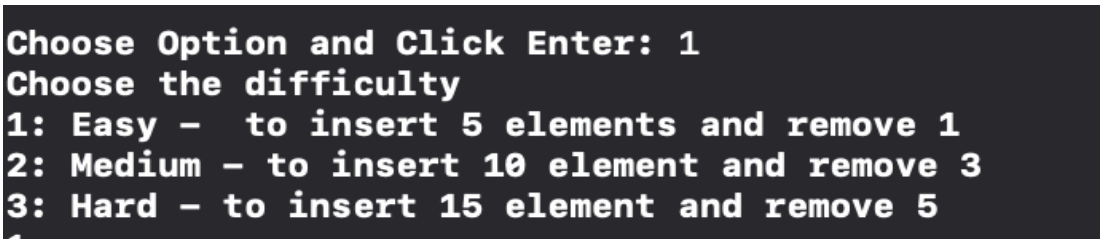
3 Интерфейс Программы

Приветственное меню с опциями на выбор:

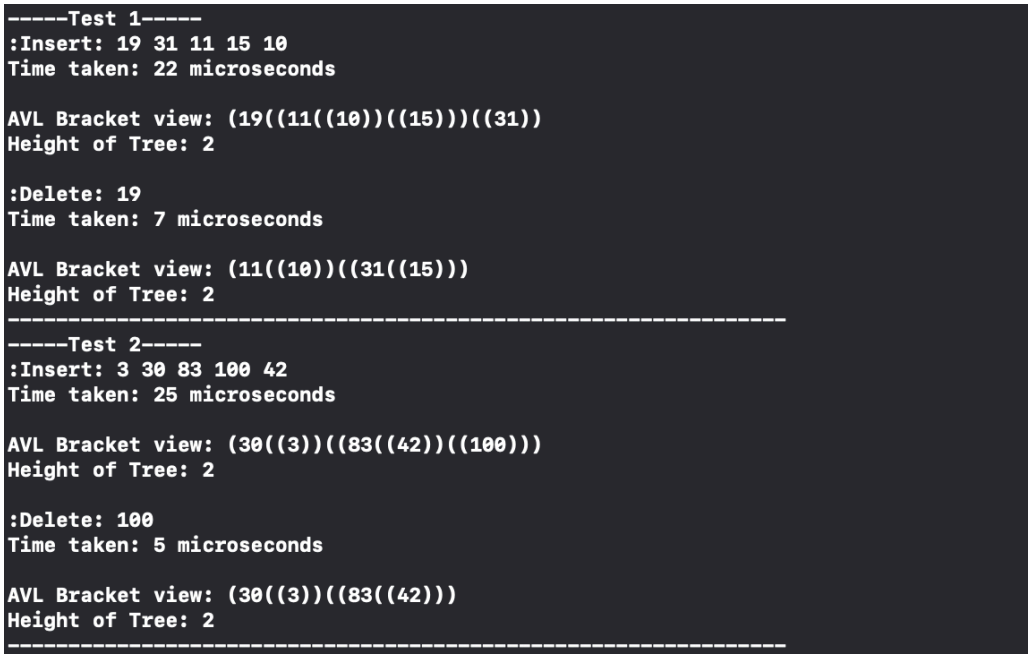


1. `Generate AVL from Random Numbers`:

   ето создает AVLTree из определенного количества случайных чисел в зависимости от уровня сложности и удаляет определенное количество случайных элементов из дерева в зависимости от уровня сложности



   Например, если мы выбираем уровень сложности 1 из меню, программа создает 5 вариантов теста.



   это два из 5 вариантов теста.. созданное дерево avl, а также время, затраченное на выполнение операций

2. `AVL from User:`

   создает Авл дерево из пользовательского ввода и элементов удаление введенного пользователем:

   Предположим, что пользователь хочет вставить 12 10 1 в дерево avl и удалить из него 10....ответ на тест приведен ниже

```
       ::::AVL TREE APP::::
::::1 Generate AVL from Random Numbers
::::2 AVL from User
::::3 AVL from Test File
::::0 Exit

Choose Option and Click Enter: 2
How many element to insert ? : 3
12
10
1
How many element to delete ? : 1
10
:Insert: 12 10 1
Time taken: 70 microseconds

AVL Bracket view: (10((1))((12))
Height of Tree: 1

:Delete: 10
Time taken: 7 microseconds

AVL Bracket view: (12((1))
Height of Tree: 1
```

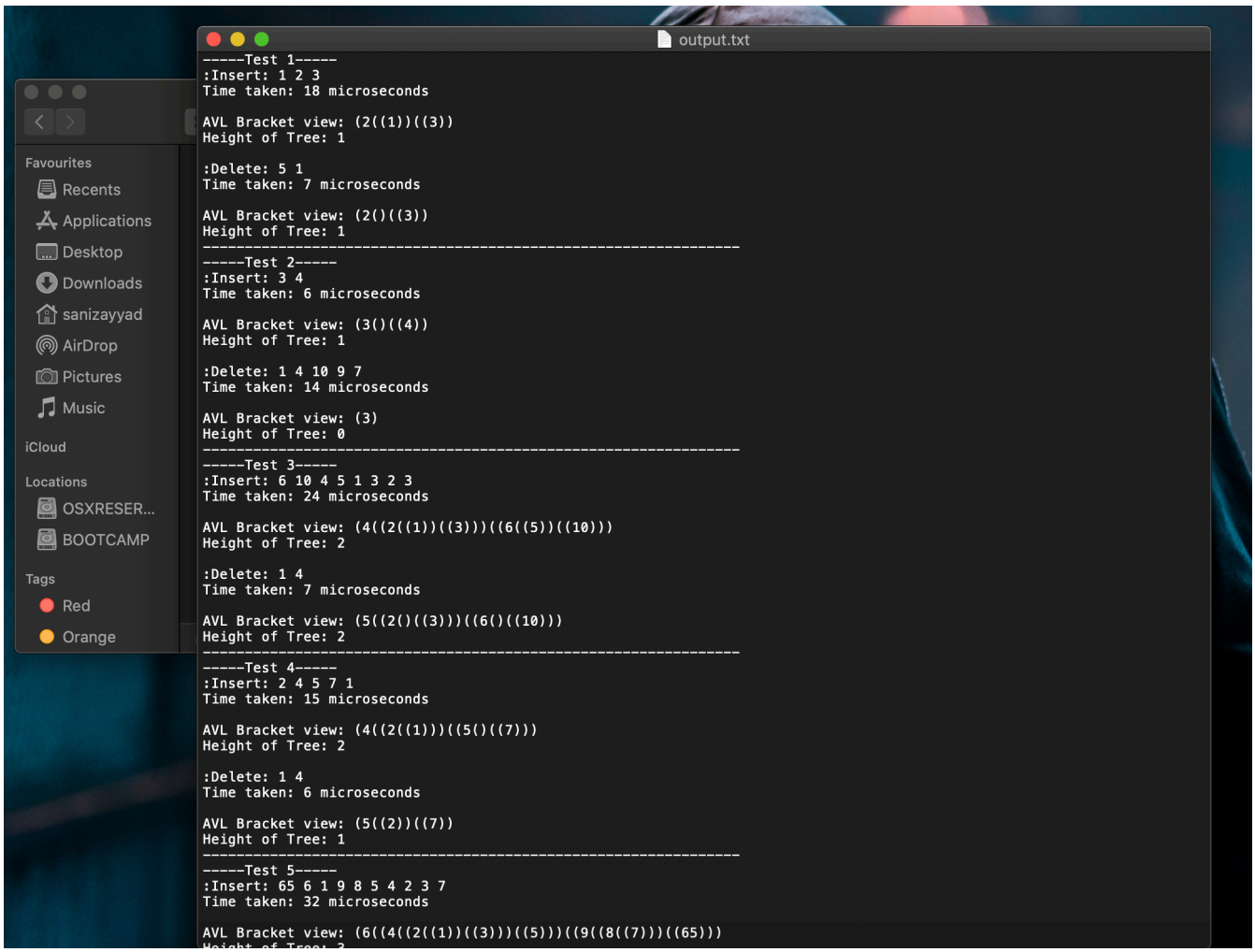
3. `AVL from Test File:`

   создает дерево Avl из предоставленного тестового файла.

```
       ::::AVL TREE APP::::
::::1 Generate AVL from Random Numbers
::::2 AVL from User
::::3 AVL from Test File
::::0 Exit

Choose Option and Click Enter: 3
> Test FilePath: /Users/sanizayyad/Documents/Sani_Zayyad/CourseWork/Test/test.txt
Reading from file:
-----Test 1-----
:Insert: 1 2 3
Time taken: 20 microseconds

AVL Bracket view: (2((1))((3))
Height of Tree: 1

:Delete: 5 1
Time taken: 8 microseconds

AVL Bracket view: (2()((3))
Height of Tree: 1
------------------------------------------------------------
```

   это пример результата теста из файла, предоставленного пользователем

После выполнения каждого задания и успешного выхода из программы, мы сохраняем все тесты в выходной файл для более детального подтверждения и получения дополнительной информации



ТЕСТЫ

Test Inputs:

1) **3 1 2 3 2**
2) **5 1 2 3 4 5 1 4**
3) **10 9 7 8 6 10 4 5 1 3 2 3 2 1 4**
4) **5 2 4 5 7 1 2 1 4**
5) **10 65 6 1 9 8 5 4 2 3 7 2 7 5**
6) **10 1 7 2 8 3 4 5 9 6 10 2 1 5**
7) **10 10 9 6 7 2 1 4 3 5 8 2 8 5**
8) **15 2 14 4 7 13 1 15 11 6 3 9 5 12 8 10 4 14 8 7 13**
9) **15 4 13 9 6 1 15 2 12 3 5 7 8 10 11 14 5 6 10 3 13 8**
10) **11 50 10 40 25 15 20 55 80 60 30 70 3 10 55 60**
11) **10 12 9 23 14 17 50 67 76 54 72 2 23 50**

Outputs:

-----Test 1-----

:Insert: 1 2 3

Time taken: 18 microseconds

AVL Bracket view: (2((1))((3))

Height of Tree: 1

:Delete: 5 1

Time taken: 7 microseconds

AVL Bracket view: (2()((3))

Height of Tree: 1

-----Test 2-----

:Insert: 3 4

Time taken: 6 microseconds

AVL Bracket view: (3()((4))

Height of Tree: 1

:Delete: 1 4 10 9 7

Time taken: 14 microseconds

AVL Bracket view: (3)

Height of Tree: 0

-----Test 3-----

:Insert: 6 10 4 5 1 3 2 3

Time taken: 24 microseconds

AVL Bracket view: (4((2((1))((3)))((6((5))((10)))

Height of Tree: 2

:Delete: 1 4

Time taken: 7 microseconds

AVL Bracket view: (5((2()((3)))((6()((10)))

Height of Tree: 2

-----Test 4-----

:Insert: 2 4 5 7 1

Time taken: 15 microseconds

AVL Bracket view: (4((2((1)))((5()((7)))

Height of Tree: 2

:Delete: 1 4

Time taken: 6 microseconds

AVL Bracket view: (5((2))((7))

Height of Tree: 1

-----Test 5-----

:Insert: 65 6 1 9 8 5 4 2 3 7

Time taken: 32 microseconds

AVL Bracket view:
(6((4((2((1))((3)))((5)))((9((8((7)))((65)))

Height of Tree: 3

:Delete: 7 5

Time taken: 8 microseconds

AVL Bracket view: (6((2((1))((4((3))))((9((8))((65)))

Height of Tree: 3

-----Test 6-----

:Insert: 1 7 2 8 3 4 5 9 6 10

Time taken: 32 microseconds

AVL Bracket view:
(3((2((1)))((7((5((4))((6)))((9((8))((10))))

Height of Tree: 3

:Delete: 1 5

Time taken: 7 microseconds

AVL Bracket view: (7((3((2))((6((4))))((9((8))((10)))

Height of Tree: 3

-----Test 7-----

:Insert: 10 9 6 7 2 1 4 3 5 8

Time taken: 30 microseconds

AVL Bracket view:
(6((2((1))((4((3))((5)))))((9((7()((8)))((10)))

Height of Tree: 3

:Delete: 8 5

Time taken: 7 microseconds

AVL Bracket view: (6((2((1))((4((3))))((9((7))((10)))

Height of Tree: 3

-----Test 8-----

:Insert: 2 14 4 7 13 1 15 11 6 3 9 5 12 8 10

Time taken: 53 microseconds

AVL Bracket view:
(7((4((2((1))((3)))((6((5)))))((13((11((9((8))((10)))((12)
))((14()((15))))

Height of Tree: 4

:Delete: 14 8 7 13

Time taken: 14 microseconds

AVL Bracket view:
(9((4((2((1))((3)))((6((5)))))((11((10))((15((12))))

Height of Tree: 3

-----Test 9-----

:Insert: 4 13 9 6 1 15 2 12 3 5 7 8 10 11 14

Time taken: 47 microseconds

AVL Bracket view:
(6((4((2((1))((3)))((5)))((9((7()((8)))((13((11((10))((12)))((15((14)))))

Height of Tree: 4

:Delete: 6 10 3 13 8

Time taken: 17 microseconds

AVL Bracket view:
(7((4((2((1)))((5)))((11((9))((14((12))((15))))

Height of Tree: 3

-----Test 10-----

:Insert: 50 10 40 25 15 20 55 80 60 30 70

Time taken: 34 microseconds

AVL Bracket view:
(50((25((15((10))((20)))((40((30))))((60((55))((80((7 0))))

Height of Tree: 3

:Delete: 10 55 60

Time taken: 11 microseconds

AVL Bracket view:
(50((25((15()((20)))((40((30))))((70()((80)))

Height of Tree: 3

-----Test 11-----

:Insert: 12 9 23 14 17 50 67 76 54 72

Time taken: 31 microseconds

AVL Bracket view:
(17((12((9))((14)))((67((50((23))((54)))((76((72))))

Height of Tree: 3

:Delete: 23 50

Time taken: 7 microseconds

AVL Bracket view:
(17((12((9))((14)))((67((54))((76((72))))

Height of Tree: 3

ЗАКЛЮЧЕНИЕ

В ходе работы была реализована avl-деревьев на языке C++ с терминальным интерфейсом, дающим множество вариантов подтверждения балансовых теорий AVl-дерева.. Кроме того, были изучены деревья AVL и их операции, а также накоплен опыт работы с ними. В ходе работы было выяснено, что операции вставки и удаления выполняются за время пропорциональное высоте дерева, т.к. в процессе выполнения этих операций производится спуск из корня к заданному узлу, и на каждом уровне выполняется некоторое фиксированное число действий.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1) Сайт ИТМО
https://neerc.ifmo.ru/wiki/index.php?title=%D0%90%D0%92%D0%9B-%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE

2) https://www.geeksforgeeks.org/ avl-tree-set-1-insertion/

3) https://www.geeksforgeeks.org/ avl-tree-set-2-deletion/

4) https://www.tutorialspoint.com/data_structures_algorithms/avl_tree_algorithm.htm

ПРИЛОЖЕНИЕ А

Code

avlTree.hpp

```
#ifndef avl_hpp
#define avl_hpp

#include <stdio.h>
#include <iostream>

struct AVLnode
{
    int key;
    int balance;
    AVLnode *left;
    AVLnode *right;

    AVLnode(int k) {
        key = k;
        balance = 0;
        left = right = nullptr;
    }
};

class AVLtree {
    public:
        AVLtree() = default;
        AVLtree(const AVLtree &) = delete;
        ~AVLtree() {}
        AVLtree &operator=(const AVLtree &) = delete;

        AVLnode *root = nullptr;
        AVLnode *insert(AVLnode *p, int k);
        AVLnode *remove(AVLnode *p, int k)  ;

        void bracketNotation(AVLnode* root,std::string& str);
        void printBaracketNotation(std::ofstream &output);
```

```
        void deleteTree(AVLnode *p);

    private:
        int height(AVLnode *p);
        void setParentBalance(AVLnode *p);

        AVLnode *rotateLeft(AVLnode *p);
        AVLnode *rotateRight(AVLnode *q);
        AVLnode *balance(AVLnode *p);
        AVLnode *findMin(AVLnode *p);
        AVLnode *removeMin(AVLnode *p);



};

#endif /* avl_hpp */

/* AVL class definition */

void AVLtree::deleteTree(AVLnode *p) {
    if (p->left) {
        deleteTree(p->left);
    }
    if (p->right) {
        deleteTree(p->right);
    }
    delete p;
}

int AVLtree::height(AVLnode *p) {
     if (p == NULL)
           return -1;
       return 1 + std::max(height(p->left), height(p->right));
}

void AVLtree::setParentBalance(AVLnode *p) {
    p->balance = height(p->right) - height(p->left);
```

```
}

AVLnode *AVLtree::insert( AVLnode *p, int k)
{
    if (p == NULL){
        return new AVLnode(k);
    }
    if (k == p->key) return p;
    else if (k < p->key)
        p->left = insert(p->left, k);
    else
        p->right = insert(p->right, k);
    return balance(p);
}

AVLnode *AVLtree::remove(AVLnode *p, int k)
{
    if (!p) return 0;
    if (k < p->key)
        p->left = remove(p->left, k);
    else if (k > p->key)
        p->right = remove(p->right, k);
    else
    {
        AVLnode *q = p->left;
        AVLnode *r = p->right;
        delete p;
        if (!r) return q;
        AVLnode *min = findMin(r);
        min->right = removeMin(r);
        min->left = q;
        return balance(min);
    }
    return balance(p);
}

AVLnode *AVLtree::findMin(AVLnode *p)
{
    return p->left ? findMin(p->left) : p;
}
```

```
AVLnode *AVLtree::removeMin(AVLnode *p)
{
    if (p->left == 0)
        return p->right;
    p->left = removeMin(p->left);
    return balance(p);
}


AVLnode *AVLtree::balance(AVLnode *p)
{
    setParentBalance(p);
    if (p->balance == 2) {
        if (p->right->balance < 0)
            p->right = rotateRight(p->right);
        return rotateLeft(p);
    }
    if (p->balance == -2) {
        if (p->left->balance > 0)
            p->left = rotateLeft(p->left);
        return rotateRight(p);
    }
    return p;
}

AVLnode *AVLtree::rotateRight(AVLnode *p)
{
    AVLnode *q = p->left;
    p->left = q->right;
    q->right = p;
    setParentBalance(p);
    setParentBalance(q);
    return q;
}


AVLnode *AVLtree::rotateLeft(AVLnode *q)
{
    AVLnode *p = q->right;
    q->right = p->left;
```

```
    p->left = q;
    setParentBalance(q);
    setParentBalance(p);
    return p;
}


void AVLtree::bracketNotation(AVLnode *root,std::string& str) {
            if (root == NULL)
                return;

            std::string s = "(";
            s += std::to_string(root->key);
            for(char i : s){
                str.push_back(i);
             }

            if (!root->left && !root->right){
                str.push_back(')');
                return;

            }

            // for left subtree
            str.push_back('(');
            bracketNotation(root->left, str);
            str.push_back(')');

            // only if right child is present to
            // avoid extra parenthesis
            if (root->right) {
                str.push_back('(');
                bracketNotation(root->right, str);
                str.push_back(')');
            }

}


void AVLtree::printBaracketNotation(std::ofstream &output){
     std::string brack;
```

```
    int h = height(root);
    bracketNotation(root,brack);
    std::cout << "\nAVL Bracket view: " << brack<< std::endl;
    std::cout << "Height of Tree: " << h << std::endl;
    output << "\nAVL Bracket view: " << brack <<"\n";
    output << "Height of Tree: " << h << "\n";

}
```

Main.cpp

```
#include <iostream>
#include <string>
#include <fstream>
#include <vector>
#include "avl.hpp"

int chooseType();
void fromRandom(const int &level,std::ofstream &output);
void fromFile(const std::string &filePath, std::ofstream &output);
void utilityAddRemove(AVLtree &avlTree, std::ofstream &output,
std::vector<int> const& vec, int const& type);
void makeAVL(AVLtree &avlTree, std::istream &file, std::ofstream &output,
int const& type);


int chooseType() {
    std::cout << "Choose the difficulty\n";
    std::cout << "1: Easy -  to insert 5 elements and remove 1 \n";
    std::cout << "2: Medium - to insert 10 element and remove 3\n";
    std::cout << "3: Hard - to insert 15 element and remove 5\n";
    int type;
    std::cin >> type;

    if ((type < 1) || (type > 3)) {
        std::cout << "You entered incorrect type\n";
        return 0;
    }
    return type;
```

```
}

void fromRandom(const int &level, std::ofstream &output){
    int array_insert_size;
    int array_delete_size;

    if(level == 1){
        array_insert_size = 5;
        array_delete_size = 1;
    }else if(level == 2){
        array_insert_size = 10;
        array_delete_size = 3;
    }else{
        array_insert_size = 15;
        array_delete_size = 5;
    }
    srand((unsigned)time(NULL));

    for (int i = 0; i < 5; i++){
        AVLtree avlTree;
        std::vector<int> inputElem;
        std::vector<int> deleteElem;
        for (int i = 0; i < array_insert_size; i++){
                int elem = rand() % 100 + 1;
                inputElem.push_back(elem);
        }
        for (int i = 0; i < array_delete_size; i++){
                int elem = rand() % array_insert_size;
                deleteElem.push_back(inputElem.at(elem));
        }
        std::cout << "-----Test " << i + 1 << "-----\n";
        output << "-----Test " << i + 1 << "-----\n";
        utilityAddRemove(avlTree, output, inputElem, 1);
        utilityAddRemove(avlTree, output, deleteElem, 2);
        std::cout <<"----------------------------------------------------
----------\n";
        output <<"----------------------------------------------------
-------\n";
        avlTree.deleteTree(avlTree.root);
    }
```

```
}

void fromFile(const std::string &filePath, std::ofstream &output){
    std::ifstream file(filePath);

    if (file.is_open()) {
        std::cout << "Reading from file:" << "\n";

        int count = 1;

        while (!file.eof()) {
               std::cout << "-----Test " << count << "-----\n";
               output << "-----Test " << count << "-----\n";
               AVLtree avlTree;
               makeAVL(avlTree,file,output,3);
               count++;
               std::cout <<"-----------------------------------------------
-----------------\n";
               output <<"-----------------------------------------------
--------------\n";
        }
    } else  std::cout << "File not opened";

}

void utilityAddRemove(AVLtree &avlTree, std::ofstream &output,
std::vector<int> const& vec, int const& type){
    if(type == 1){
         std::cout<<":Insert: ";
         output<<":Insert: ";
         for (auto i = vec.begin(); i != vec.end(); ++i){
             avlTree.root =  avlTree.insert(avlTree.root, *i);
             std::cout<< *i << " ";
             output<< *i << " ";

         }
    }else{
        std::cout<<"\n:Delete: ";
```

```
        output<<"\n:Delete: ";
        for (auto i = vec.begin(); i != vec.end(); ++i){
            avlTree.root =  avlTree.remove(avlTree.root, *i);
            std::cout<< *i << " ";
            output<< *i << " ";
        }
    }
    avlTree.printBaracketNotation(output);
}

void makeAVL(AVLtree &avlTree, std::istream &file, std::ofstream &output,
int const& type){
    int array_insert_size;
    int array_delete_size;
    std::vector<int> inputElem;
    std::vector<int> deleteElem;

    if(type == 1){
        int difficulty = chooseType();
        if(difficulty != 0)
            fromRandom(difficulty,output);

    }else if (type == 2){
        std::cout<< "How many element to insert ? : ";
        std::cin >> array_insert_size;
        for (int i = 0; i < array_insert_size; ++i) {
            int elem;
            std::cin >> elem;
            inputElem.push_back(elem);
        }
        std::cout<< "How many element to delete ? : ";
        std::cin >> array_delete_size;

        for (int i = 0; i < array_delete_size; ++i) {
            int elem;
            std::cin >> elem;
            deleteElem.push_back(elem);
        }

    }else{
```

```
            file >> array_insert_size;
            for (int i = 0; i < array_insert_size; ++i) {
                int elem;
                file >> elem;
                inputElem.push_back(elem);
            }

            file >> array_delete_size;
            for (int i = 0; i < array_delete_size; ++i) {
                int elem;
                file >> elem;
                deleteElem.push_back(elem);
            }
    }

    if(type!=1){
        utilityAddRemove(avlTree, output, inputElem, 1);
        utilityAddRemove(avlTree, output, deleteElem, 2);
        avlTree.deleteTree(avlTree.root);
    }
}

int main(){
    int command;
    std::string ouputfile;
    std::cout << "> Output FilePath: ";
    std::cin >> ouputfile;
    std::ofstream output(ouputfile, std::ios_base::app);

    do{
        std::cout<<"\n\n\t\t::::AVL TREE APP::::"<<std::endl;
        std::cout<<"::::1 Generate AVL from Random Numbers"<<std::endl;
        std::cout<<"::::2 AVL from User"<<std::endl;
        std::cout<<"::::3 AVL from Test File "<<std::endl;
        std::cout<<"::::0 Exit"<<std::endl;

        std::cout<<"\nChoose Option and Click Enter: ";
        std::cin >> command;
        AVLtree avlTree;
```

```
        switch(command)
                {
                    case 0:
                        break;
                    case 1:
                           makeAVL(avlTree, std::cin,output, 1);
                           break;
                    case 2:
                           makeAVL(avlTree, std::cin,output, 2);
                            break;
                    case 3:
                        {
                            std::string filePath ;
                            std::cout << "> Test FilePath: ";
                            std::cin >> filePath;
                            fromFile(filePath,output);
                            break;
                        }
                    default:
                        std::cout<<"Sorry! wrong input"<<std::endl;
                        break;
                }
        }while(command != 0);


    return 0;
}
```